

Practical 'The Bioinformatics Lab'

Compile, Make, Text-Editors

Daniel Bader

Lecturer: Dr. Laszlo Kajan

Overview

- Compile Programs
 - Java
 - C/C++
- Deploy Programs
 - Make
 - Make Imitators
- Archives and Packages
 - tar
 - gzip
- Text Editors

Compiler or Interpreter?

- Compiler
 - Transformation of source (programming) language
→ target (machine) language → execution
 - First complete compiler: 1957 FORTRAN team at IBM
 - Development of programs that are machine-independent
- Interpreter
 - 1) executes the source code directly
 - 2) source code → intermediate code
→ immediately executes (Perl, Python,...)
 - 3) explicitly executes stored precompiled code made by a compiler
which is part of the interpreter system

Structure of Compiler

- Front-end
 - Determining the correctness of the syntax of programs
 - Error reports
- Middle-end
 - Removal of useless or unreachable code
 - discovery and propagation of constant values
- Back-end
 - translating intermediate representation from middle-end into assembly code
 - most algorithms for optimization are NP → heuristic techniques

[illegible]

Language 1 source code

Compiler front-end for language 1

```
graph TD
    A[Lexical Analyzer (Scanner)] --> B[Syntax/Semantic  
Analyzer (Parser)]
    B --> C[Intermediate-code  
Generator]
```

Non-optimized intermediate code

```
public class OddEven {
    private int input;
    public OddEven() {
        input = parseInt(Integer);
    }
    public void calculate() {
        if (input % 2 == 0)
            System.out.println("Even");
        else
            System.out.println("Odd");
    }
    public void main(String args[]) {
    }
}
```

Language 2 source code

Compiler front-end for language 2

```
graph TD; A[Lexical Analyzer (Scanner)] --> B[Syntax/Semantic Analyzer (Parser)]; B --> C[Intermediate-code Generator];
```

Non-optimized intermediate code

Intermediate code optimizer

Optimized intermediate code

Target-1 Code Generator

|Target-1 machine code



Target-2
Code Generator

|Target-2 machine code



Compiling Java

- Write Java program Hello.java

```
public class Hello {  
    public static void main (String[] args) {  
        System.out.println("Hello, World");  
    }  
}
```

- Compile HelloWorld.java

```
javac Hello.java → Hello.class
```

- Pack to JAR archive

```
jar cmf Hello.mf Hello.jar Hello.class Hello.java
```

- Execute:

- `java -jar Hello.jar`
- `java Hello`

Compiling C/C++

- Write C program main.c

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    printf("Hello World\n");
    return(0);
}
```

- Compile main.c

```
gcc main.c -o HelloWorld
```

- Execute:

```
./HelloWorlds
```

Deploy Programs: Make

- Usage:
 - Compile programs
 - Integrate programs into system
- Repeated Usage:

Determination of program parts that need to be recompiled
- Initialization file (e.g. „make.ini“) is read first, followed by the makefile
- make.ini → customization
- Architecture independent

Structure of Makefile

- Components:
macros, directives, rules, dependency and shell lines
- Example:

```
project.exe : main.obj io.obj
    tlink c0s main.obj io.obj, project.exe,, cs /Lf:\bc\lib

main.obj : main.c
    bcc -ms -c main.c

io.obj : io.c
    bcc -ms -c io.c
```

Make Imitators

- Perl
 - Module::Build
create a Build.PL
 - ExtUtils::MakeMaker
create a Makefile.PL
- Python
 - setup() within distutils.core module
 - “The basic do-everything function that does most everything you could ever ask for from a Distutils method.”

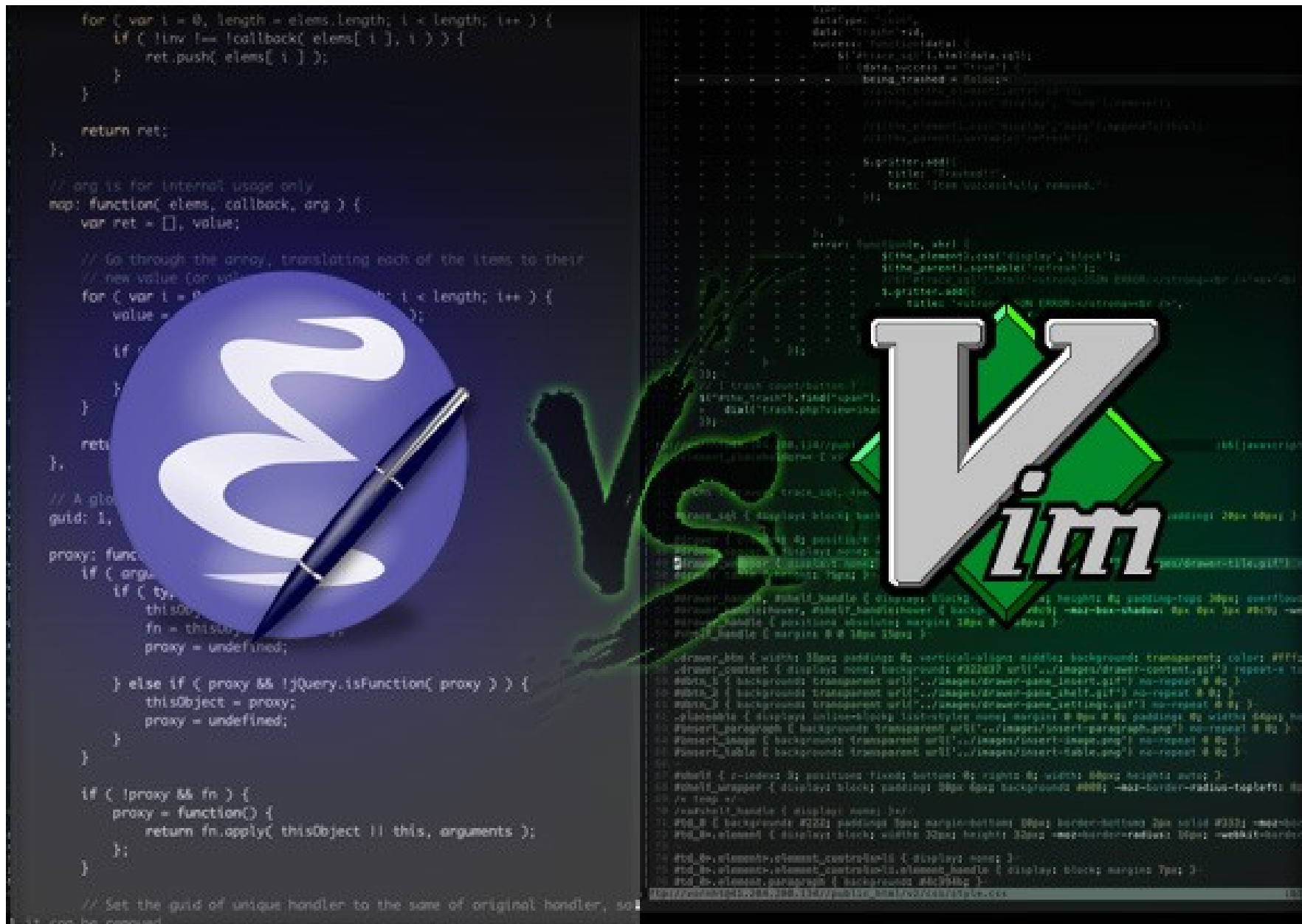
Packaging

- General: Storing multiple files in one archive-file
- Tarball package
 - archives created by `tar` utility
 - packing: `tar cvf package.tar package`
 - unpacking: `tar xvf package.tar`
 - often additional compression with `gzip` tool
- Often used for Software distribution
- Content

README, configure, makefile, code

Staged Installation

- Install package and copy products to a staged directory
- Install in an user defined directory
- DESTDIR is a variable prefixed to each installed target file
 - DESTDIR variable specified by user with the make parameters
 - example: `make DESTDIR=/tmp/stage install`



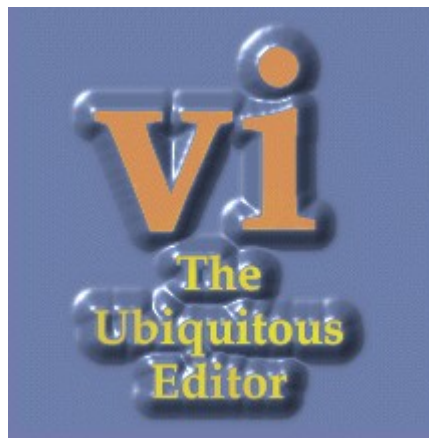
Terminal-Based Text Editors

- Diakonos
 - standard key bindings (Ctrl-C to copy, Ctrl-V to paste, ...)
 - dependent on Ruby
- Jed
 - small, fast (faster startup than bash)
 - drop-down menus
- Nano
 - Ubuntu's default terminal editor
 - Clone of Pico, the editor of the Pine email client

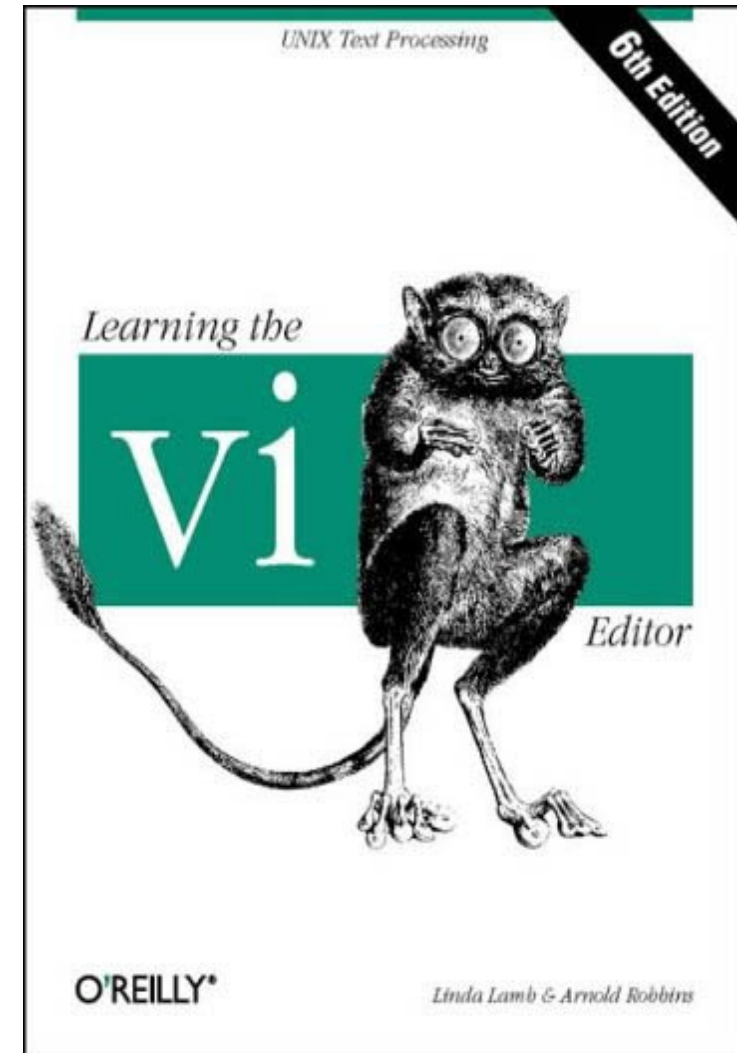
Emacs



- Initial release 1976 by Richard Stallman
- Developer: The GNU project
- Written in C and Emacs LISP
- first program released by the nascent GNU project
- Handling files up to 512 Mb



- Initial release 1976 by Bill Joy
- Written in C
- 2009 Linux Journal readers:
 - vi most widely used text editor (36%)
 - beating gedit, the second (19%)
- Derivatives
 - Elvis written by Steve Kirkendall; first to provide color syntax highlighting
 - Vim "Vi IMproved"





- Initial release 1991 by Bram Moolenaar
- Written in C and vimscript
- graphical user interface gVim
- 2006: most popular editor (Linux Journal readers)
- Some of Vim's enhancements:
 - completion
 - comparison and merging of files (vimdiff)
 - multiple level undo history, persist across editing sessions

References

- Compiler
 - Wikipedia
 - http://www.eis.mdx.ac.uk/staffpages/r_bornat/books/compiling.pdf
- Make
 - <http://www.opussoftware.com/tutorial/TutMakefile.htm>
 - http://www.mathcs.richmond.edu/~lbarnett/MCS_dept/compiling.pdf
 - Wikipedia
- Perl
 - <http://search.cpan.org/~leont/Module-Build-0.40/lib/Module/Build.pm>
 - <http://learnperl.scratchcomputing.com/tutorials/configuration/>

References

- Jar, Tar, Gzip
man pages
- Text Editors
 - <http://www.linuxlinks.com/article/20080824052425167/Editors.html>
 - Wikipedia

Thank you for your Attention!